Simple Geometric Relationships in Game Programming

Note to self — a collection of simple geometric ideas worth exploring through small game-programming experiments.

- A lot of useful game geometry can be obtained **without explicitly storing geometric information**.
- Game objects already provide positions, and relationships between those positions contain considerable geometry.
- Two sprite positions A and B immediately provide:

- Difference:

$$D = B - A$$

- Distance:

$$|D| = \sqrt{D_x^2 + D_y^2}$$

- Direction:

$$\theta = \operatorname{atan2}(D_y, D_x)$$

- Unit direction:

$$\hat{D} = \frac{D}{|D|}$$

- Perpendicular/normal:

$$N = (-D_y, D_x)$$

- Midpoint:

$$M = \frac{A + B}{2}$$

- The difference vector is especially important:

- It describes both **direction and displacement**.

- It can be scanned using:

$$P(t) = A + t(B - A)$$

for $0 \leq t \leq 1$.

• A simple `for` loop can therefore visit every approximate position along the segment.

• This can be used for collision testing, trails, beams, ropes, interpolation, effects and ray-like operations.

• The normal of a pair of points is surprisingly useful:

• It gives the direction **sideways to the relationship** between two objects.
• It can position objects beside a road or path.
• It can create the two sides of a road or ribbon.
• It can place effects beside a line connecting two sprites.
• It can provide left/right avoidance directions.

• It can define a local coordinate system for the pair.

• Combining midpoint and normal is a particularly simple construction:

$$C = M + w\hat{N}$$

gives a point a distance w to one side of the midpoint.

• $M + w\hat{N}$ and $M - w\hat{N}$ give symmetric positions on either side.

• This can be useful for formations, labels, effects, weapons, obstacles and decorative geometry.

• Two points can therefore define much more than a line:

• position
• displacement
• distance
• heading
• tangent
• normal
• midpoint
• local coordinate frame
• corridor
• beam
• ribbon

• boundary

• Pair-wise geometry is particularly natural in games because pairs arise automatically:

• last position → current position
• current road point → next road point
• player → enemy
• projectile → target
• two cars
• two mouse positions
• two waypoints
• two collision points

• two vertices of a polygon

• Consecutive positions contain **motion geometry**:

$$D = P_{\text{current}} - P_{\text{previous}}.$$

• D gives movement direction.

• $|D|$ gives distance travelled per frame.
• Dividing by frame time gives velocity.
• A perpendicular gives sideways movement.

• Comparing successive displacement vectors gives turning information.

• Three positions provide even more:

$$D_1 = P_1 - P_0, \qquad D_2 = P_2 - P_1.$$

• Dot product:

$$D_1 \cdot D_2$$

measures whether motion is continuing in roughly the same direction.

• 2D cross product:

$$D_{1x}D_{2y} - D_{1y}D_{2x}$$

determines the direction of the turn.

• The angle between them describes how sharply a path turns.

• Repeated scanning of these relationships can estimate curvature.

• A sequence of points can therefore be treated as a **geometric signal**:

$$P_0, P_1, P_2, \ldots, P_n$$

from which a loop can extract:

• direction

• speed
• normals
• angles
• curvature
• turning direction

• local coordinate frames

• This is closely related to old-school game graphics:

• road-racing games can construct road edges from centreline points and their normals;
• Doom-like renderers can use slopes, angles, projection and line relationships;
• sprites can be sheared, oriented and positioned using simple vector relationships;

• scanlines can turn geometric equations into very cheap loops.

• A useful general question when programming a game is therefore:

> **What geometry is already implicit in the relationships between the positions I have?**

• Rather than adding more stored state, first look for geometry that can be **derived from pairs or sequences of existing points**.

• A useful experimental pattern:

$$\boxed{\text{existing positions} \rightarrow \text{difference} \rightarrow \text{derived geometry} \rightarrow \text{simple loop} \rightarrow \text{game effect}}$$

• The deeper attraction is that elementary vector algebra can act as a kind of **geometry generator**. Very small equations can produce useful spatial structures when repeatedly evaluated across points, pixels, road segments, sprites or frames.